Licenciatura em Engenharia Informática e Computadores

- Segurança Informática-

19/20 SI

1ª Série de Exercícios -> Resolução Grupo 6 LI51N

Docente: José Simão

Alunos: Luís Guerra nº 43755

David Albuquerque nº43566

Hugo Almeida nº 42931

1.

Um esquema assimétrico de assinatura digital e um esquema MAC têm como semelhanças a garantia de autenticação da mensagem enviada, ambos tiram partido de um verificador determinístico, ambos os esquemas envolvem o envio da mensagem por um meio inseguro, bem como uma marca para a autenticação da mensagem por um canal seguro.

As diferenças entre os dois são que enquanto no esquema MAC a chave é a mesma para o emissor e o recetor, tendo que ser partilhada, no esquema de assinatura de digital existem 2 chaves, a chave pública e privada do emissor, sendo partilhada esta chave publica ao recetor.

Os critérios de decisão a ponderar nesta decisão são, a vulnerabilidade da partilha da chave única do esquema MAC, a complexidade da produção e verificação que assinatura digital do esquema assimétrico exige. A limitação constante de só poder usar este esquema de assinatura digital.

2.

É computacionalmente fazível que, dado m, obter m' != m tal que H(m') = H(m), devido ao facto de a função de hash depender dos dois últimos blocos da mensagem, sendo perfeitamente fazível em tempo útil devido ao facto de ser possível, com blocos iniciais diferentes, obter os 2 últimos blocos finais, gerando a mesma funçao hash.

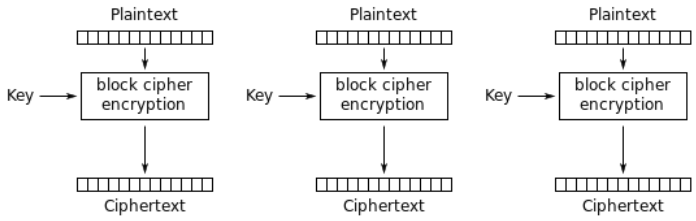
3.

Visto que estamos a lidar com cifra assimétrica, e que o algoritmo de cifra será determinístico e não probabilístico sabemos $E_a(k)(x) = E_a(k)(y)$, portanto basta o atacante saber o x ou y que chegará facilmente á chave (k) desencriptando assim a mensagem

4.

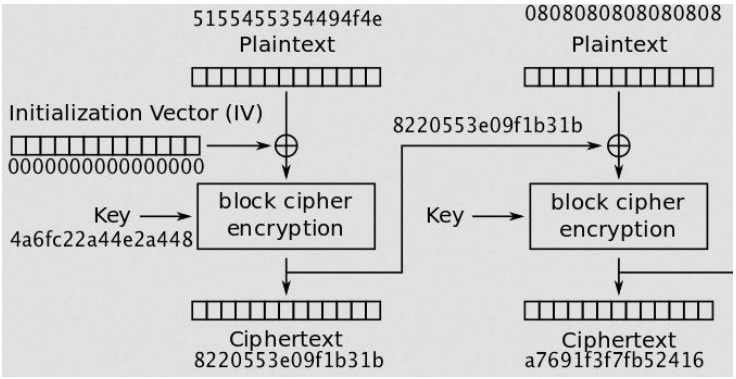
Admitindo o uso de chaves aleatórias tanto para A (chaves de 56 bits) e para B (chaves de 128 bits), podemos afirmar que há possibilidade de um criptograma produzido pelo sistema A ser mais difícil de criptoanalisar do que um criptograma do sistema B dependendo do modo de operação usada pelas primitivas.

Teoricamente um criptograma encriptado com a primitiva AES seria mais difícil de criptoanalisar visto que a sua chave tem mais bits, mas se o AES utilizar o modo de operação ECB e o DES utilizar o modo de operação CBC, o DES será muito mais fácil de criptoanalisar.



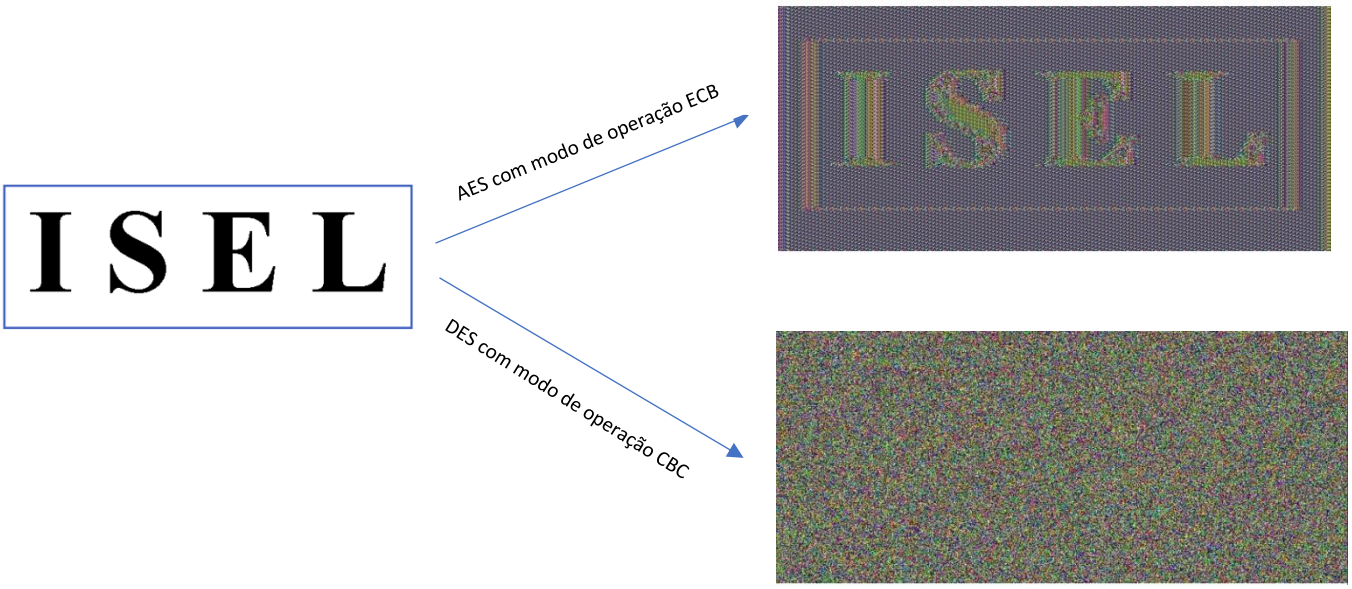
Electronic Codebook (ECB) mode encryption

e.g DES com modo de operação ECB



e.g AES com modo de operação CBC

Exemplo prático:

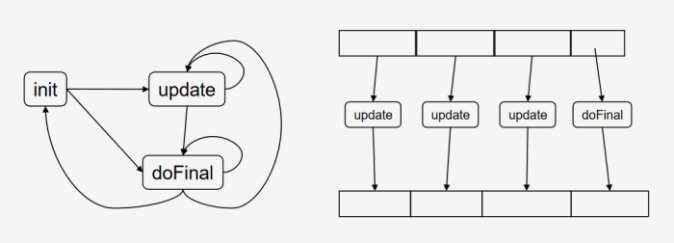


5.

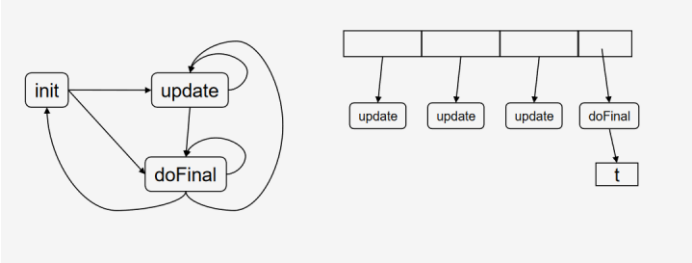
Na biblioteca *Java Cryptography Architecture* (JCA) existem algumas engine classes como a classe Cipher, Signature e MAC. Estas classes disponibilizam métodos que possibilitam a aplicação incremental das proteções associadas ás respetivas classes.

Na classe Cipher temos o método init que recebe como parâmetros o modo (cifra, decifra, wrap ou unwrap), chave, parâmetros específicos do algoritmo e gerador aleatório.

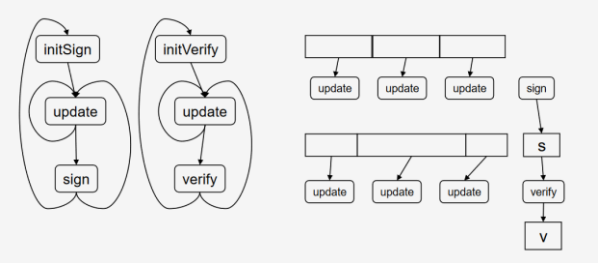
Após o uso deste método temos os métodos de cifra update (recebe parte da mensagem e retorna parte do criptograma) e o método doFinal (que recebe o final da mensagem e retorna o final do criptograma). O método update é a operação incremental que vai recebendo partes da mensagem e retorna o criptograma dessa mensagem (byte [] -> byte []).



Na classe MAC também temos o método init que recebe a chave e alguns parâmetros específicos do algoritmo. Após o uso desse método temos os métodos de geração de marca update (recebe parte da mensagem e não retorna nada) e o método doFinal (recebe o final da mensagem e retorna a marca). Como na classe Cipher , aqui o método update é a operação incremental que vai recebendo a mensagem(byte [] -> void) . No fim, é que o método doFinal retorna a respetiva marca.



Na classe Signature como para além da geração temos também a verificação vão existir 2 tipos de métodos para cada etapa. Na parte de inicialização temos os métodos initSign que recebe a chave privada e um gerador aleatório, e o método initVerify que recebe a chave pública. Após o uso desses métodos temos os métodos de geração, como nas outras classes, só que neste caso é geração de assinatura que são o update que recebe parte da mensagem (byte [] -> void) e o método sign que finaliza a operação incremental retornando a assinatura (void -> byte []). Para além destes métodos temos também os métodos de verificação da assinatura update que realiza o mesmo que o método anterior de mesmo nome e o método verify que finaliza a operação incremental retornando a validade da assinatura (void -> {true,false}.



A vantagem do uso de aplicações incrementais neste caso é no caso em que a mensagem seja um ficheiro ou até uma mensagem “grande” e assim é repartida e enviada e cifrada por partes facilitando a encriptação da mensagem. Outro caso é quando recebemos só partes da mensagem e não a mensagem completa e assim permite-nos ir cifrando conforme se vai recebendo a informação.

6.

6.1

No âmbito de garantir essa autenticidade é necessário que as funções de hash geradas do lado do emissor da mensagem e do recetor sejam as mesmas, sabemos que a assinatura gerada com base nesse hash e realizada com a chave pública nunca foi alterada o que faz com que o certificado seja autêntico e seguro.

6.2

No âmbito de validação do certificado de folha C, são usadas as chaves públicas dos certificados intermédios e não as chaves privadas dos mesmos. As chaves privadas são utilizadas única e exclusivamente na geração de assinaturas.

7.

7.1

Para realizarmos iremos encriptar um ficheiro e comparar o tamanho do ficheiro original com o ficheiro novo criado.

```
OpenSSL> enc -des-ede-cbc -in f1.txt -out f12.bin
```

No caso do modo de operação CBC o ficheiro original continha 6 bytes de informação e o ficheiro novo criado tem 24 bytes onde podemos concluir que este modo tem padding pois o valor do tamanho do ficheiro criado é múltiplo de 16.


```
OpenSSL> enc -des-ecb -in f1.txt -out f12.bin
```

No caso do modo de operação ECB acontece o mesmo que o modo anterior, ou seja, o ficheiro original tem 6 bytes e o criado tem 24 bytes logo tem padding.

```
OpenSSL> enc -des-cfb -in f1.txt -out f12.bin
```

No caso do modo de operação CBF acontece que o tamanho do ficheiro criado não é múltiplo de 16, ou seja, o ficheiro original tem 6 bytes e o criado tem 22 bytes logo não tem padding.

"f1.txt" selected (6 bytes) "f12.bin" selected (22 bytes)

```
OpenSSL> enc -des-ede-ofb -in f1.txt -out f12.bin
```

No caso do modo de operação OFB acontece o mesmo que o modo de operação CBF, ou seja, o ficheiro original tem 6 bytes e o criado tem 22 bytes logo não tem padding.

7.2

Para este exercício começamos pela primeira etapa, criação de 3 ficheiros com 5, 10 e 16 bytes respetivamente.

```
[10/21/19]seed@VM:~$ echo -n "12345" > f1.txt
[10/21/19]seed@VM:~$ echo -n "1234567891" > f2.txt
[10/21/19]seed@VM:~$ echo -n "1234567891234567" > f3.txt
```

Depois passamos à encriptação dos mesmos através do comando disponível no enunciado.

```
OpenSSL> enc -aes-128-cbc -e -in f1.txt -out f1cypher.bin
OpenSSL> enc -aes-128-cbc -e -in f2.txt -out f2cypher.bin
OpenSSL> enc -aes-128-cbc -e -in f3.txt -out f3cypher.bin
```

Passada esta etapa verificou-se que os tamanhos dos ficheiros cifrados criados divergem dos ficheiros originais como era de esperar.

"f1.txt" selected (5 bytes) "f1cypher.bin" selected (32 bytes)
"f2.txt" selected (10 bytes) "f2cypher.bin" selected (32 bytes)
"f3.txt" selected (16 bytes) "f3cypher.bin" selected (48 bytes)

Agora vamos desencriptar os ficheiros criados e tentar olhar para o padding que foi adicionado a cada ficheiro usando o comando apresentado no enunciado.

```
OpenSSL> enc -aes-128-cbc -d -nopad -in f1cypher.bin -out f1decyp
her.txt
```

12345

Carateres de padding (f1decypher.txt)

```
[10/21/19]seed@VM:~$ hexdump -C f1decypher.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |1234
5...........|
00000010
```

```
[10/21/19]seed@VM:~$ xxd f1decypher.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345.........
```

```
OpenSSL> enc -aes-128-cbc -d -nopad -in f2cypher.bin -out f2decyp
her.txt
```

Carateres de padding (f2decypher.txt)

```
[10/21/19]seed@VM:~$ hexdump -C f2decypher.txt
00000000  31 32 33 34 35 36 37 38  39 31 06 06 06 06 06 06  |1234
567891......|
00000010
```

```
[10/21/19]seed@VM:~$ xxd f2decypher.txt
00000000: 3132 3334 3536 3738 3931 0606 0606 0606  1234567891....
..
```

```
OpenSSL> enc -aes-128-cbc -d -nopad -in f3cypher.bin -out f3decyp
her.txt
```

Carateres de padding (f3decypher.txt)

```
[10/21/19]seed@VM:~$ hexdump -C f3decypher.txt
00000000  31 32 33 34 35 36 37 38  39 31 32 33 34 35 36 37  |1234
567891234567|
00000010  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  |....
............|
00000020
```

```
[10/21/19]seed@VM:~$ xxd f3decypher.txt
00000000: 3132 3334 3536 3738 3931 3233 3435 3637  12345678912345
67
00000010: 1010 1010 1010 1010 1010 1010 1010 1010  ................
..
```